Online Retailer

Introduction

The online retailer application is a CLI-based application that serves as a secure, feature-rich platform designed to facilitate e-commerce transactions for both customers and administrators. Users can easily create accounts, browse available products, and place orders, all within a streamlined and user-friendly interface. Customers can also view past orders. Their personal information is kept safe through secure authentication practices such as password hashing. The system provides clear, organised data on products, orders, and account management, making it an efficient solution for both regular and admin users.

Security

Security is a key feature in the system, designed to protect both user data and system integrity. The application employs multiple layers of security, starting with encrypted password storage using hashing techniques to prevent unauthorised access. Role-based access control ensures only administrators can perform sensitive actions, such as viewing all users or managing orders. Additionally, the application is fortified against common cyber-attacks like SQL injection, brute force attempts, and denial-of-service (DoS) attacks, ensuring that hackers cannot easily exploit vulnerabilities. A crucial component of cybersecurity is logs. A logging mechanism tracks critical actions, enabling system audits and helping detect suspicious activities in real-time (Mezmo, N.D.). These comprehensive security measures ensure that users' personal data and the overall system remain safe from internal and external threats.

Features By Role

User

1. Account Creation: Users can create an account by registering with a unique username and password. Passwords are hashed before storage for security.
2. Login: Users can log in to the system using their credentials.
3. View Products: Users can view all available products and their prices in the system.
4. Place Orders: Users can place orders for products, specifying the quantity.
5. View Orders: Users can view their order history.
6. Logout: Users can log out of the system, ending their session.

Admin

1. Account Creation: Admins can create an account by registering with a unique username and password. Passwords are hashed before storage for security.
2. Login: Admins can log in to the system using their credentials.
3. Manage Users: Admins can view a list of all registered users, along with their roles (admin or regular user). They can also permanently delete any user.
4. Delete Users: Admins can delete users from the system.
5. Manage Products: Admins can add, update, or remove products from the system.

6. View Orders: Admins can view all orders placed in the system.
7. Toggle Security: Admins can toggle security settings to enable or disable enhanced security features for testing purposes. Security is ON by default.
8. View Logs: Admins can view system logs to monitor critical actions taken by users (admin or regular), including login attempts, order placements, and admin actions for auditing purposes.
9. Logout: Admins can log out of the system, ending their session.

Hacker (simulated for testing)

1. Brute Force Attack: Hackers may attempt brute force attacks on login credentials.
2. Denial of Service (DoS): Hackers may attempt a DoS attack to overwhelm the system with unexpected traffic.
3. API Injection Attack: Hackers may attempt API injection to create an admin user.

Tools and Technologies

1. Programming Language: Python
2. Database: SQLite
3. REST API Framework: Flask
4. Testing Framework: Pytest
5. Linter: Pylint

Implementation Details

Database

SQLite is distinguished by its exceptional efficiency, serverless nature, and self-contained structure, making it notable for its simplicity and ease of integration (GeeksforGeeks, 2024). Therefore, the application uses SQLite as the database for managing users, products, and orders. A singleton Database class ensures that there is a single connection throughout the application lifecycle. Default users and products are pre-populated to get started.

Below are the user credentials:

1. Username - admin, Password - admin, Role - Admin
2. Username - test, Password - test123, Role - Regular User

Below are the product details:

1. Product - Laptop, Price - $1,000
2. Product - Smartphone, Price - $700

User Management

The application supports two types of users: Admin and Regular Users. User-related operations are handled by the UserManager class. Some of the operations require having admin privileges. Admin-only operations impacting the overall system are handled by the AdminManager class.

Security Management

Passwords are hashed using ‘bcrypt’ before storing them in the database. All actions regardless of their nature are logged and available in the app.log file. Admins have a dedicated menu option to access these system logs. Moreover, admins can toggle security (ON by default) to simulate cybersecurity attacks, examine the system’s response and strengthen the system’s security defences.

Cybersecurity Attacks Simulation

The HackerManager class is designed to simulate various types of cybersecurity attacks on the online retailer application. Its primary purpose is to test the system's resilience against potential hacking attempts, including brute-force attacks, Denial of Service (DoS) attacks, and API injection attacks. This ensures the system is prepared for real-world cyberattacks by proactively identifying potential vulnerabilities.

Order and Product Management

The product-related operations are managed by the ProductManager class whereas order-related operations are managed by the OrderManager class. Products in the system can be viewed by all user types however they can be managed only by admins. Regular users can place orders and view order history whereas admins can view all orders of the system.

REST API Management

The application exposes certain functionality through APIs, allowing external systems to interact with it programmatically. One key API is the Create User API, which allows an external service to create users in the application. Given its lightweight nature, ease of adoption, comprehensive documentation, and widespread popularity, Flask is an excellent choice for the development of RESTful APIs. APIs are managed by API class (Fox, 2023).

Menu Management

Upon launching the application, the MenuManager class displays the main menu with various options which serves as the entry point for users to interact with the system. The menu dynamically adjusts based on the user’s role. The user selects an option by inputting a corresponding number. The menu manager ensures the input is valid and prompts the user to select again if invalid. If valid, it proceeds to execute the appropriate function.

Error Handling

The application makes extensive use of Python’s try-except blocks to catch exceptions that might occur and prevent the application from crashing due to unhandled exceptions. User inputs are validated to prevent errors before they occur. When the application exits, a cleanup is performed. Error messages are simple and informative, ensuring non-technical users understand what went wrong.

Test Coverage

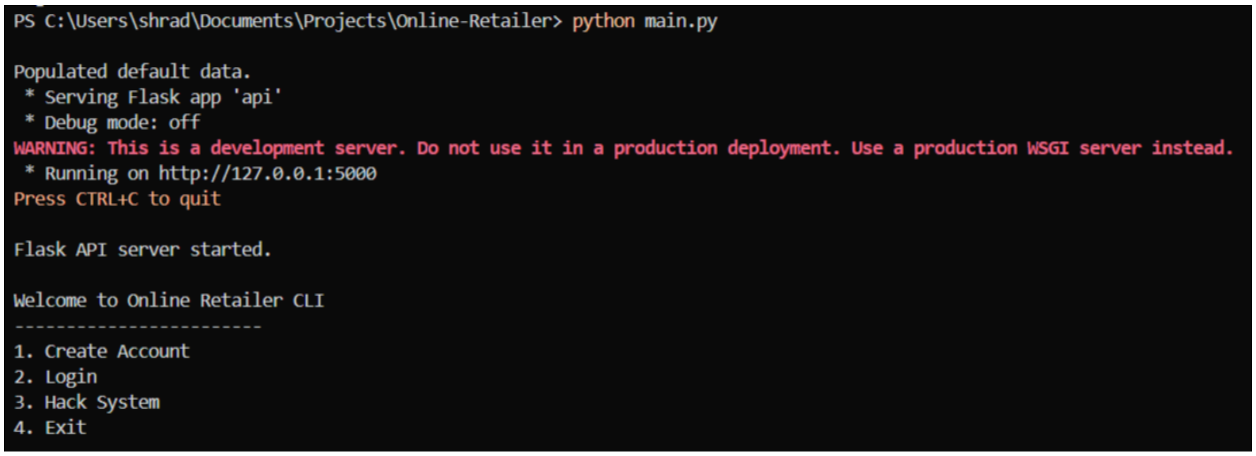
The application includes automated testing using the ‘pytest’ framework. The tests cover core areas of application like user, product, and order management. The tests are available in the tests directory.

Future Enhancements

1. Ability to promote/demote a user.
2. Ability to update the password.
3. Ability to lock the account after a certain no. of failed login attempts.

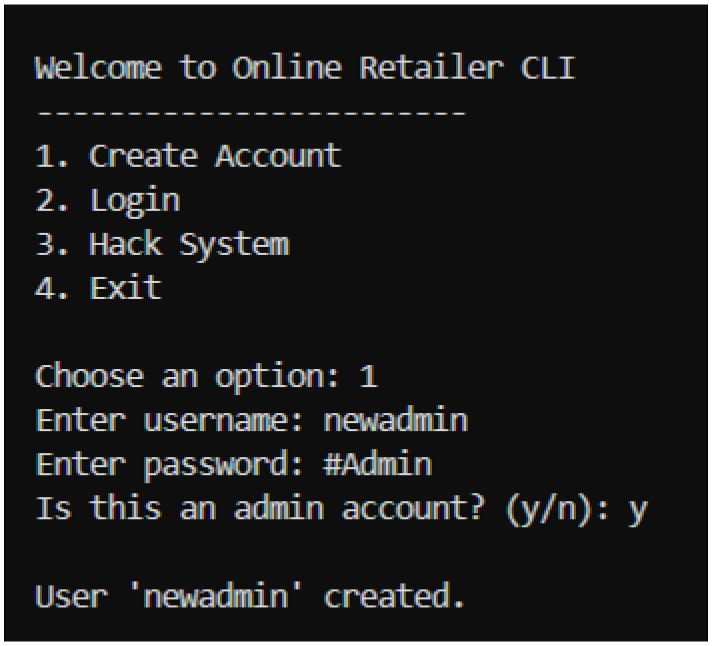
Working Evidence

Start Application



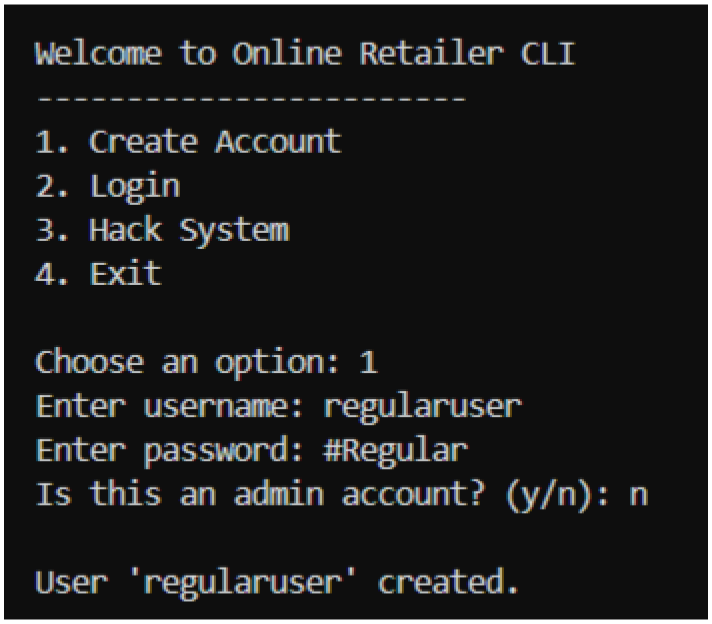
```
PS C:\Users\shrad\Documents\Projects\Online-Retailer> python main.py

Populated default data.
 * Serving Flask app 'api'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit

Flask API server started.

Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit
```

Create Account (Admin User)



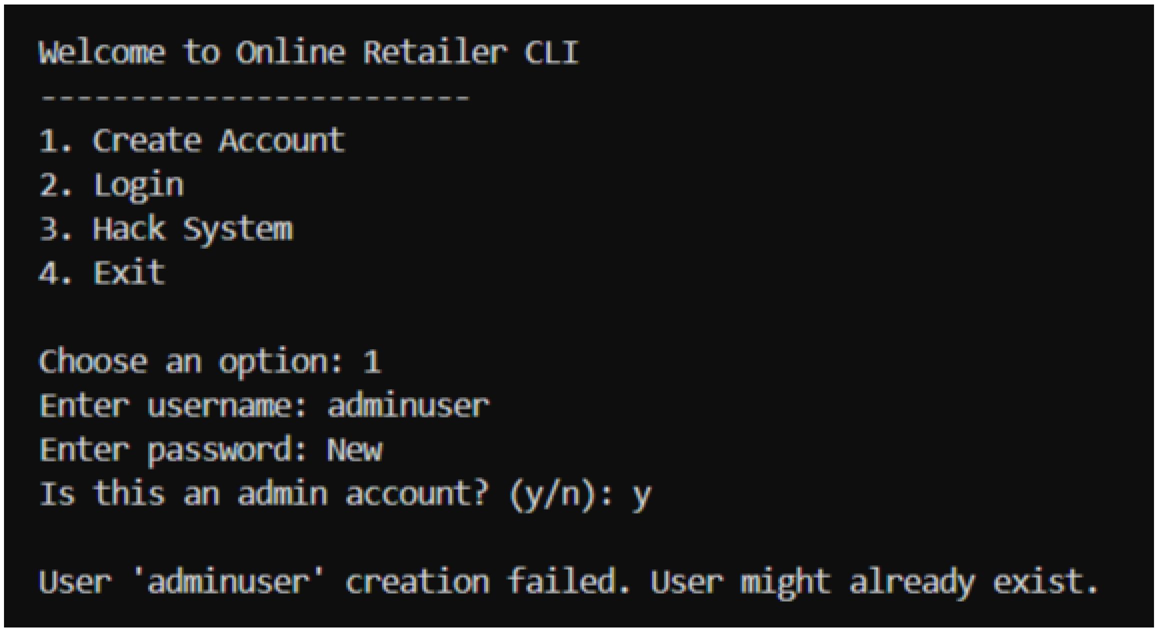
```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 1
Enter username: newadmin
Enter password: #Admin
Is this an admin account? (y/n): y

User 'newadmin' created.
```

Create Account (Regular User)



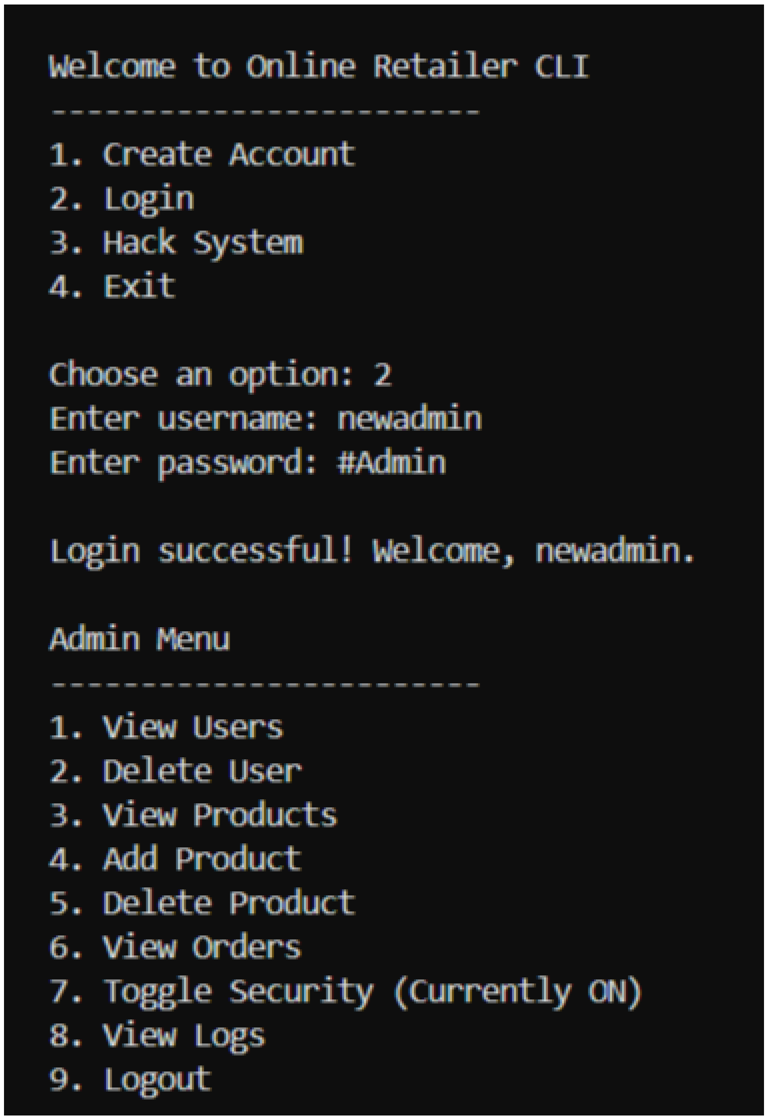
```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 1
Enter username: regularuser
Enter password: #Regular
Is this an admin account? (y/n): n

User 'regularuser' created.
```

Create Account (Existing User)



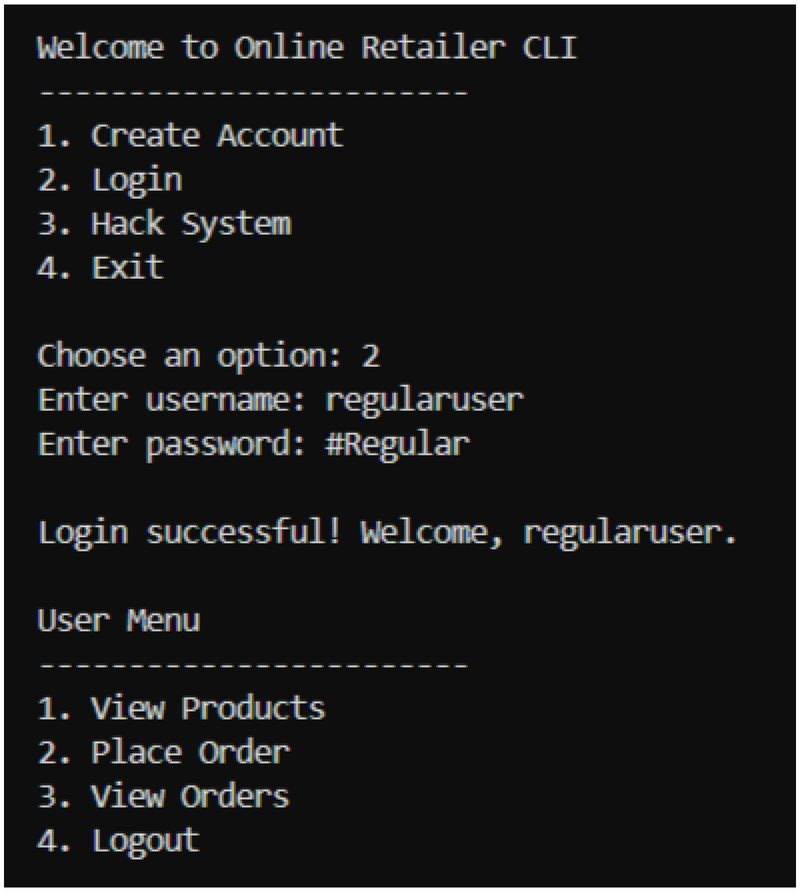
```
Welcome to Online Retailer CLI
-------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 1
Enter username: adminuser
Enter password: New
Is this an admin account? (y/n): y

User 'adminuser' creation failed. User might already exist.
```

Login (Admin User)



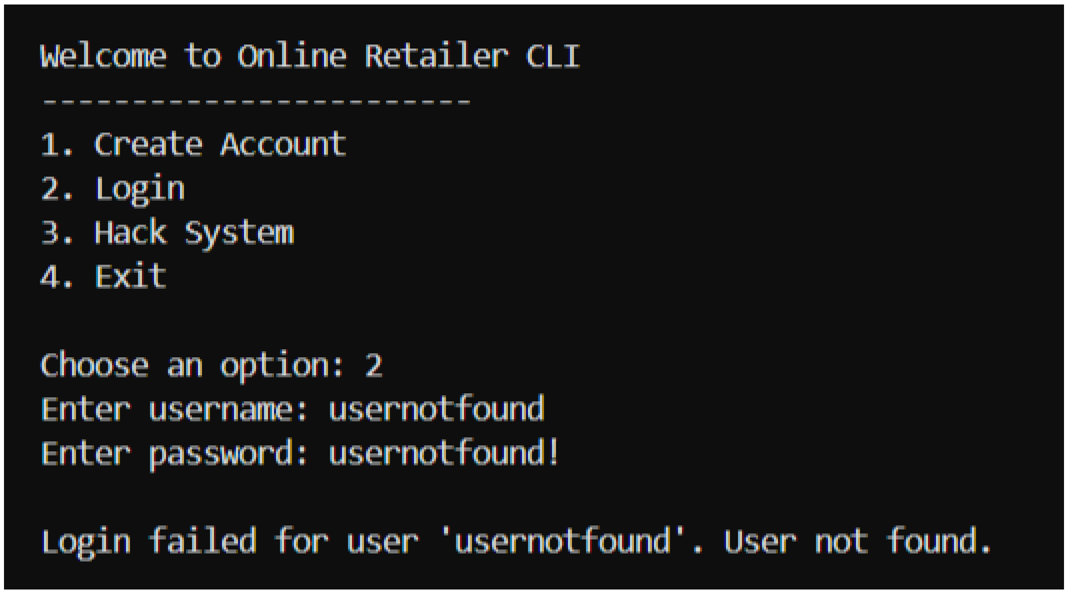
```
Welcome to Online Retailer CLI
-------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 2
Enter username: newadmin
Enter password: #Admin

Login successful! Welcome, newadmin.

Admin Menu
-------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout
```

Login (Regular User)

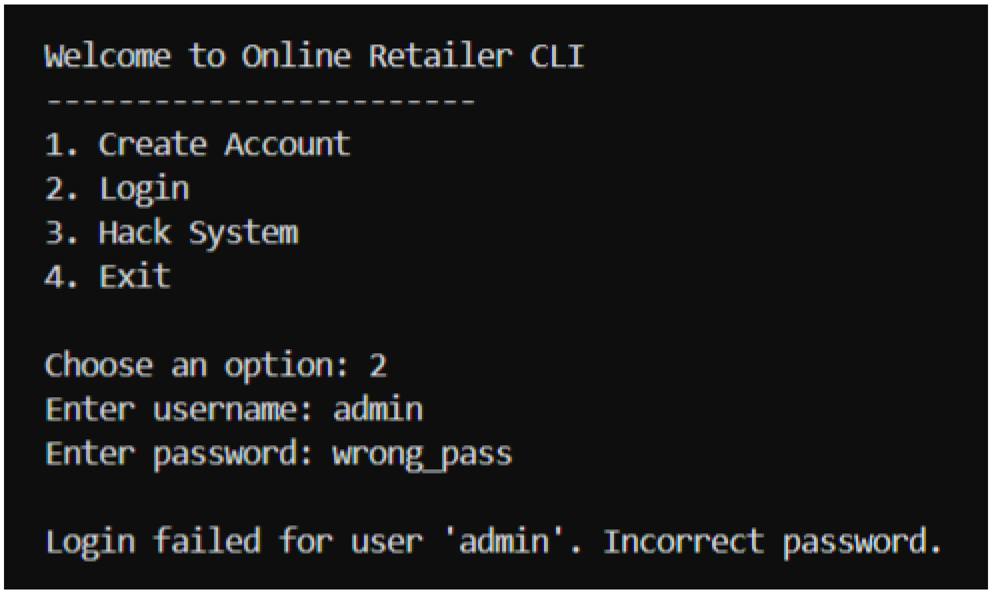


```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 2
Enter username: regularuser
Enter password: #Regular

Login successful! Welcome, regularuser.

User Menu
------------------------
1. View Products
2. Place Order
3. View Orders
4. Logout
```

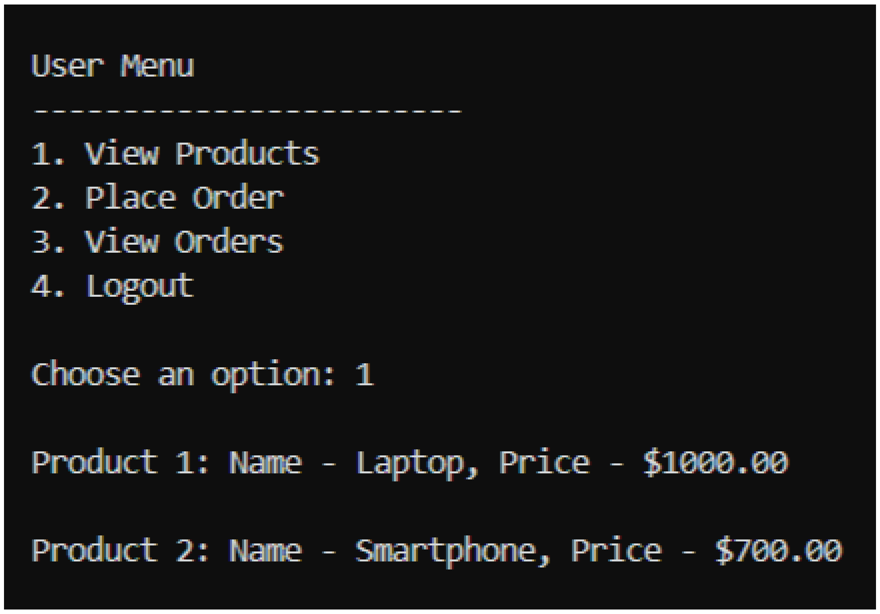
Login (Non-existent User)


```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 2
Enter username: usernotfound
Enter password: usernotfound!

Login failed for user 'usernotfound'. User not found.
```

Login (Incorrect Password)

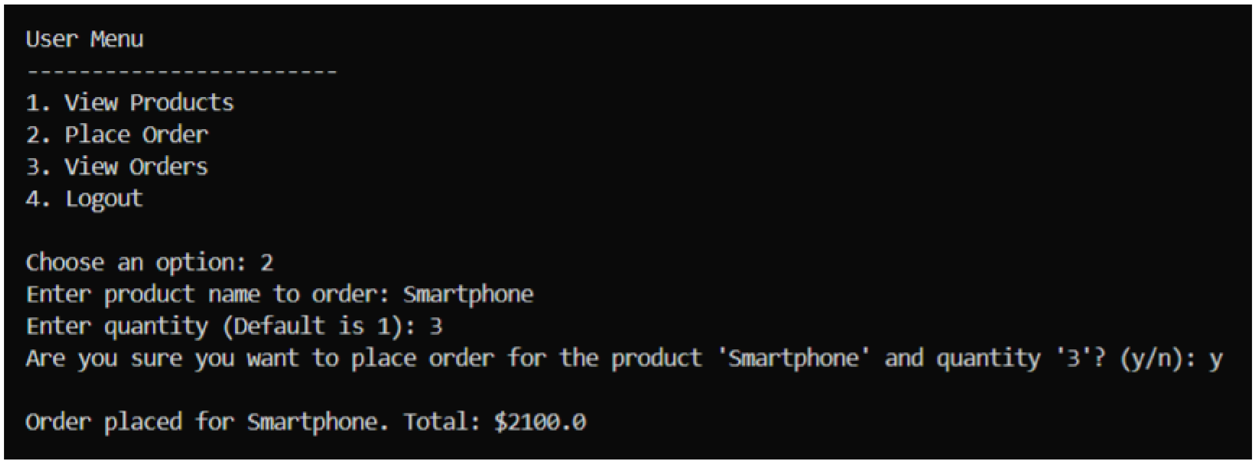

```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 2
Enter username: admin
Enter password: wrong_pass

Login failed for user 'admin'. Incorrect password.
```

View Products (Regular User)



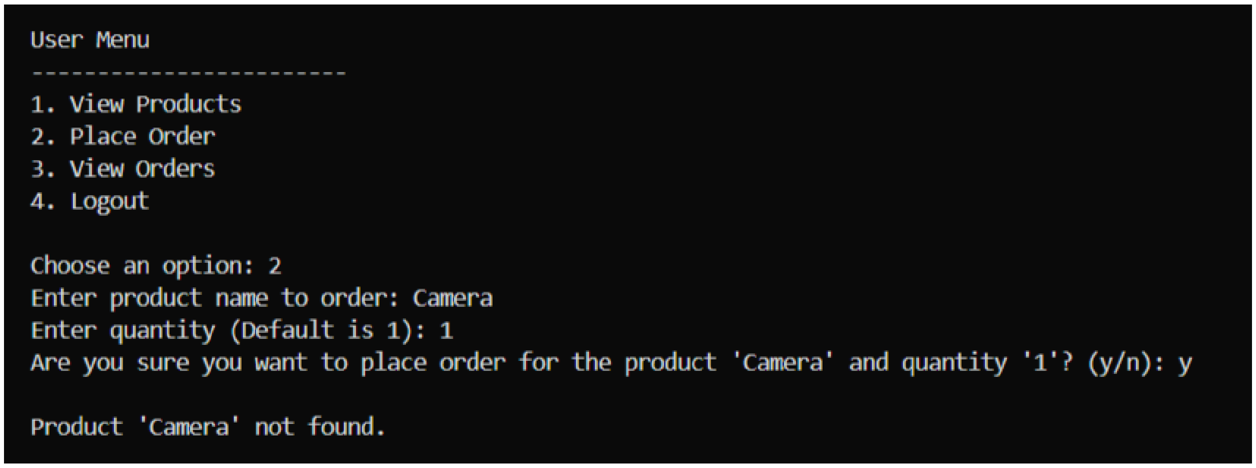
```
User Menu
------------------------
1. View Products
2. Place Order
3. View Orders
4. Logout

Choose an option: 1

Product 1: Name - Laptop, Price - $1000.00

Product 2: Name - Smartphone, Price - $700.00
```

Place Order



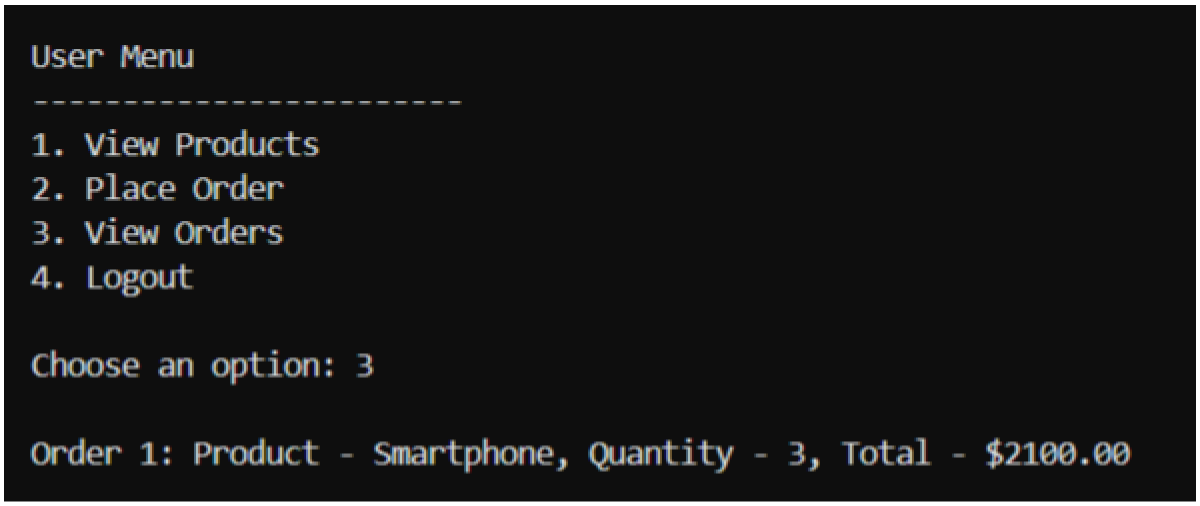
```
User Menu
------------------------
1. View Products
2. Place Order
3. View Orders
4. Logout

Choose an option: 2
Enter product name to order: Smartphone
Enter quantity (Default is 1): 3
Are you sure you want to place order for the product 'Smartphone' and quantity '3'? (y/n): y

Order placed for Smartphone. Total: $2100.0
```

Place Order (Non-existent Product)



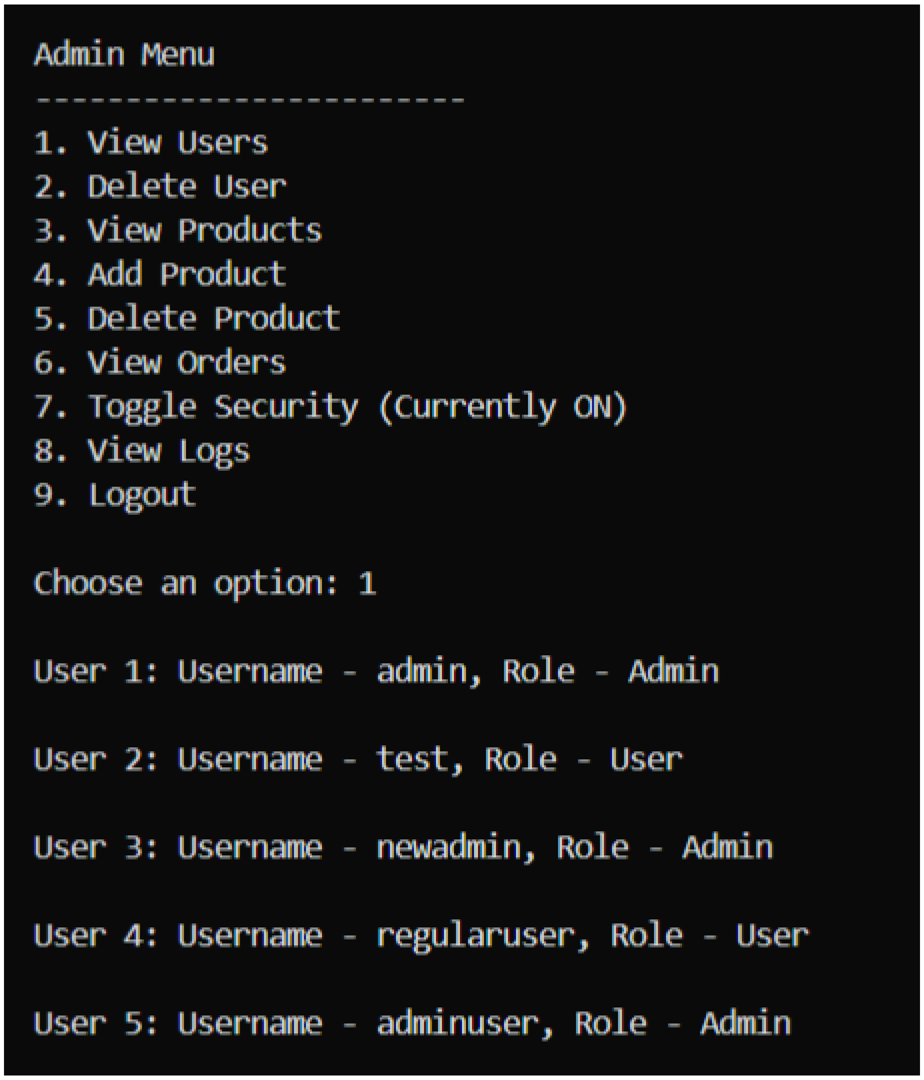
```
User Menu
------------------------
1. View Products
2. Place Order
3. View Orders
4. Logout

Choose an option: 2
Enter product name to order: Camera
Enter quantity (Default is 1): 1
Are you sure you want to place order for the product 'Camera' and quantity '1'? (y/n): y

Product 'Camera' not found.
```

View Orders (Regular User)



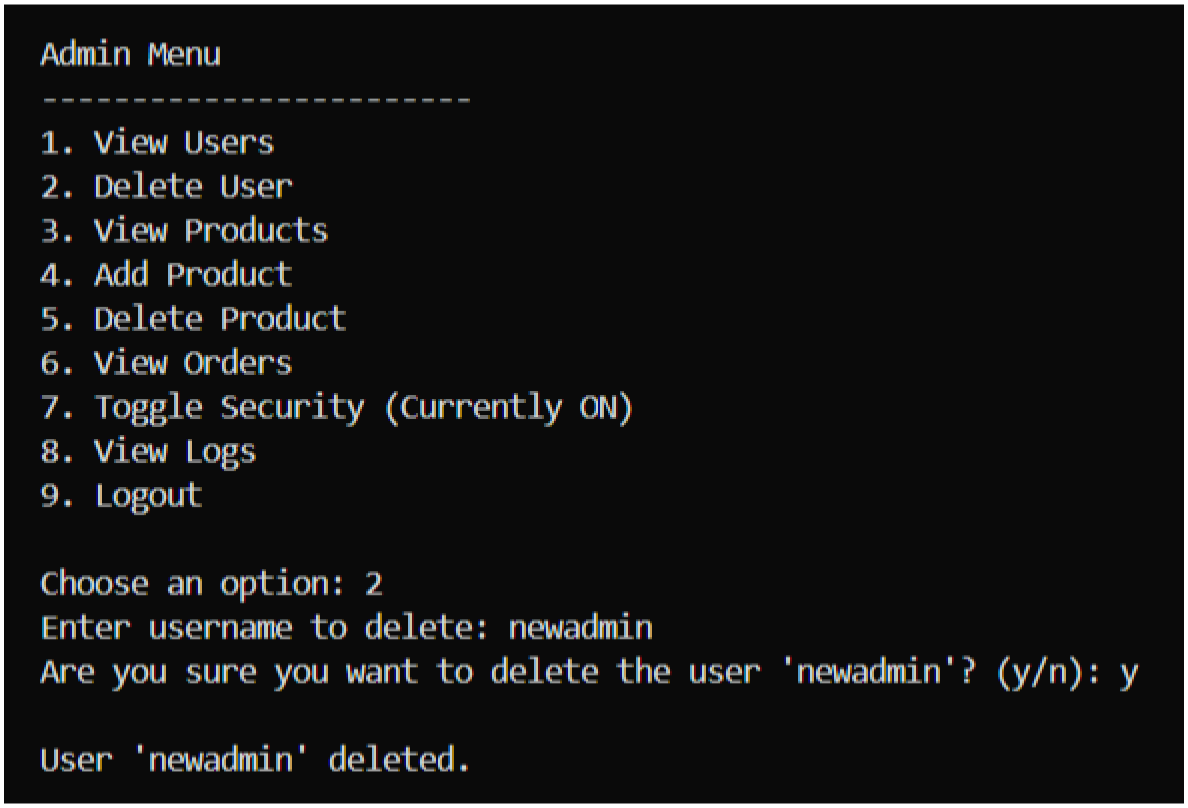
```
User Menu
------------------------
1. View Products
2. Place Order
3. View Orders
4. Logout

Choose an option: 3

Order 1: Product - Smartphone, Quantity - 3, Total - $2100.00
```

View Users

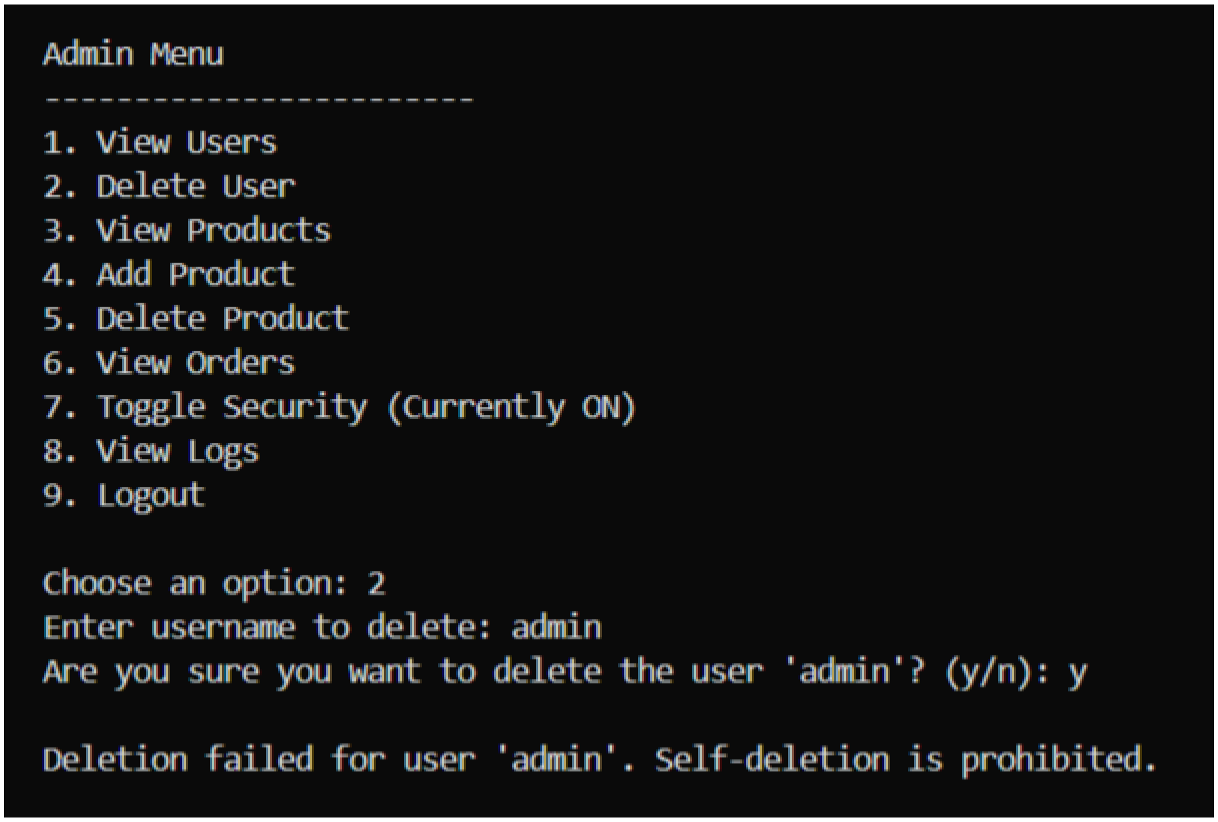


```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 1

User 1: Username - admin, Role - Admin

User 2: Username - test, Role - User

User 3: Username - newadmin, Role - Admin

User 4: Username - regularuser, Role - User

User 5: Username - adminuser, Role - Admin
```

Delete User

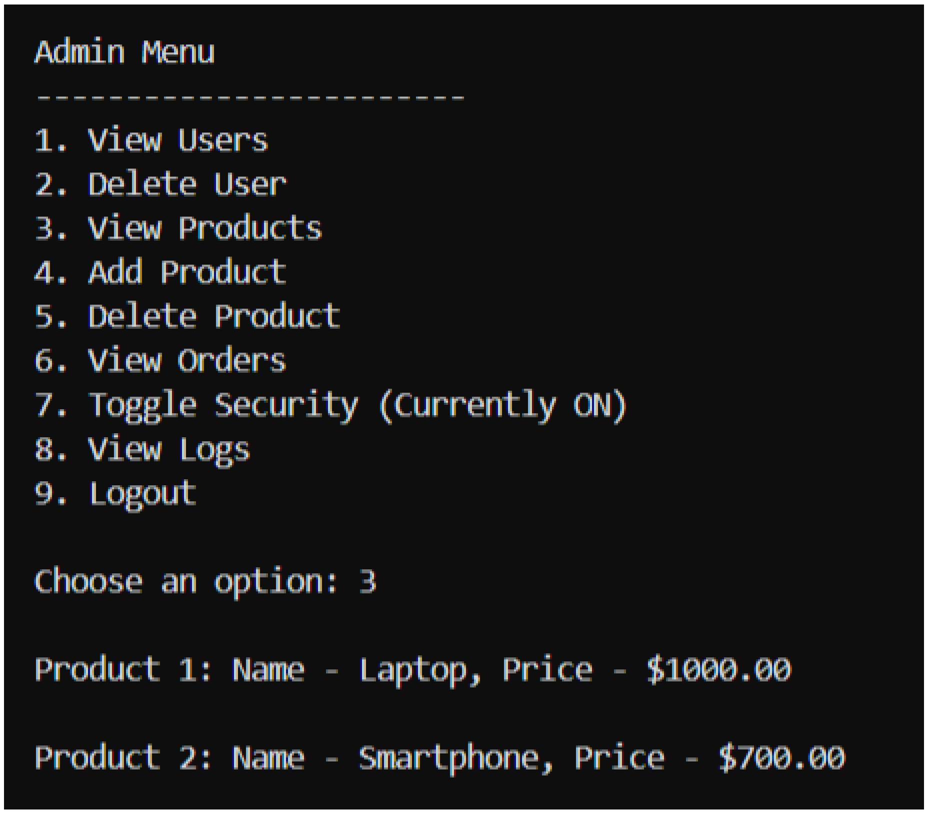

```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 2
Enter username to delete: newadmin
Are you sure you want to delete the user 'newadmin'? (y/n): y

User 'newadmin' deleted.
```

Delete User (Self)



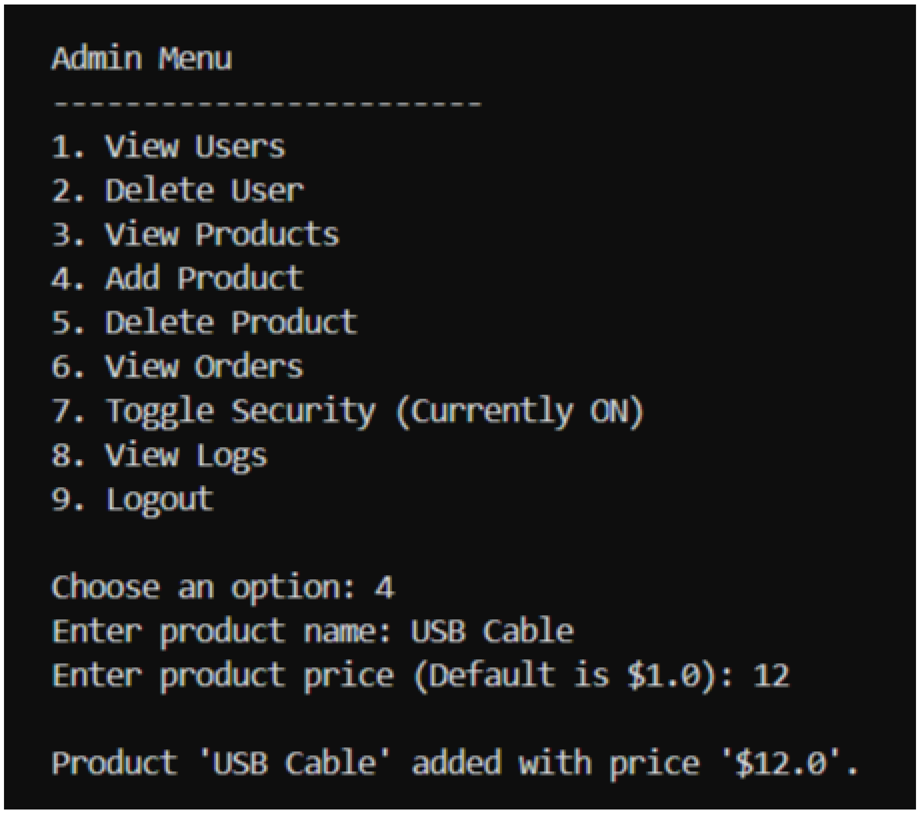
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 2
Enter username to delete: admin
Are you sure you want to delete the user 'admin'? (y/n): y

Deletion failed for user 'admin'. Self-deletion is prohibited.
```

View Products (Admin User)



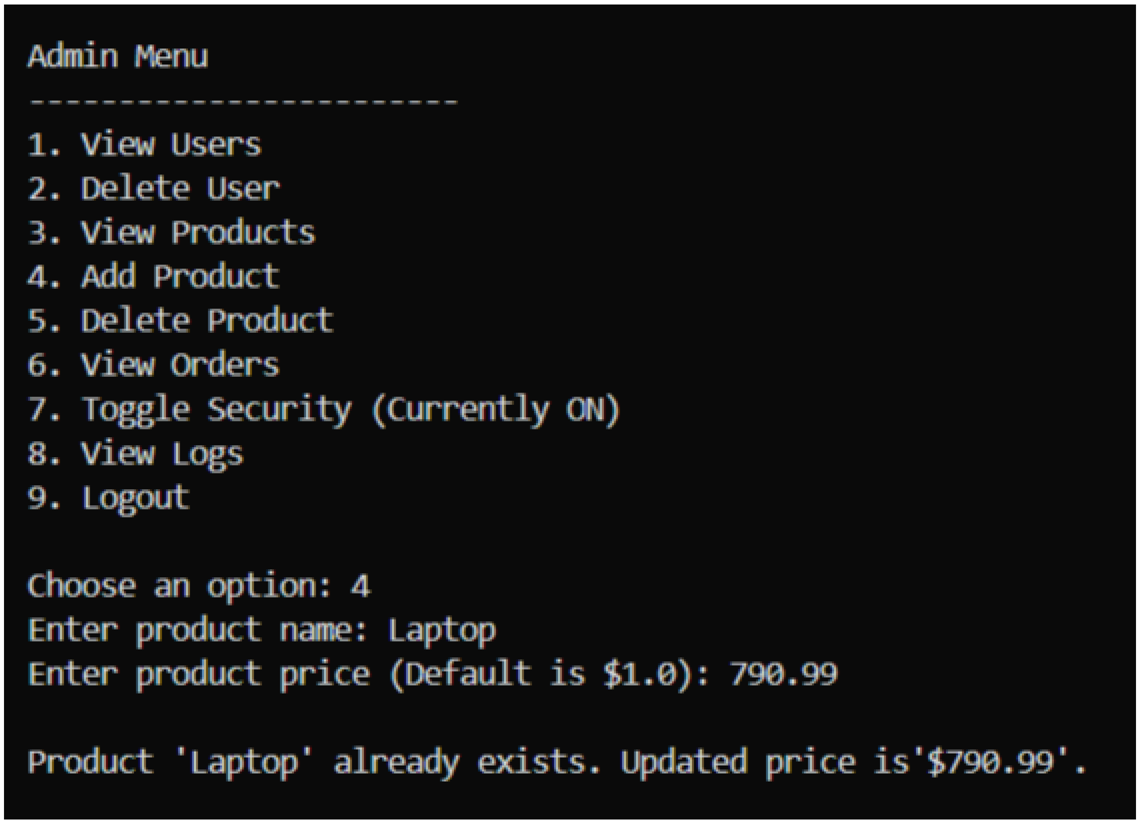
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 3

Product 1: Name - Laptop, Price - $1000.00

Product 2: Name - Smartphone, Price - $700.00
```

Add Product



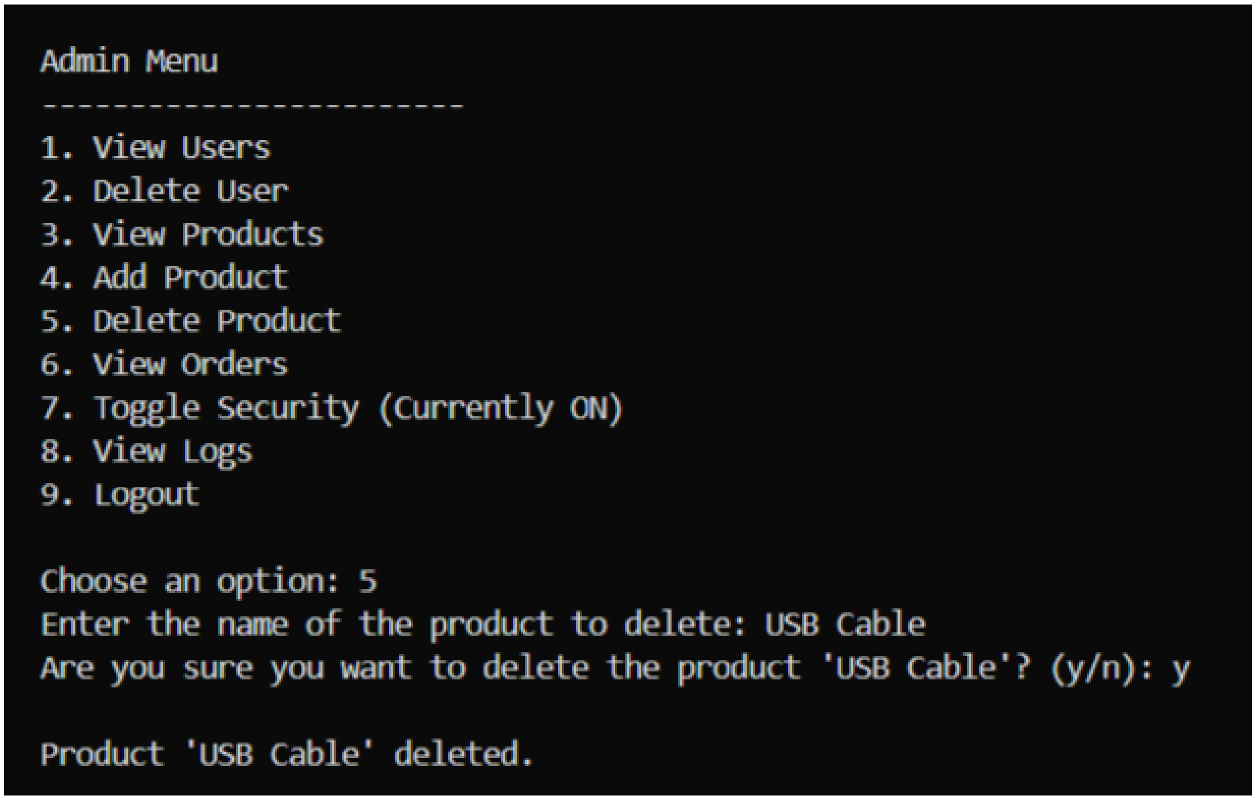
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 4
Enter product name: USB Cable
Enter product price (Default is $1.0): 12

Product 'USB Cable' added with price '$12.0'.
```

Update Product



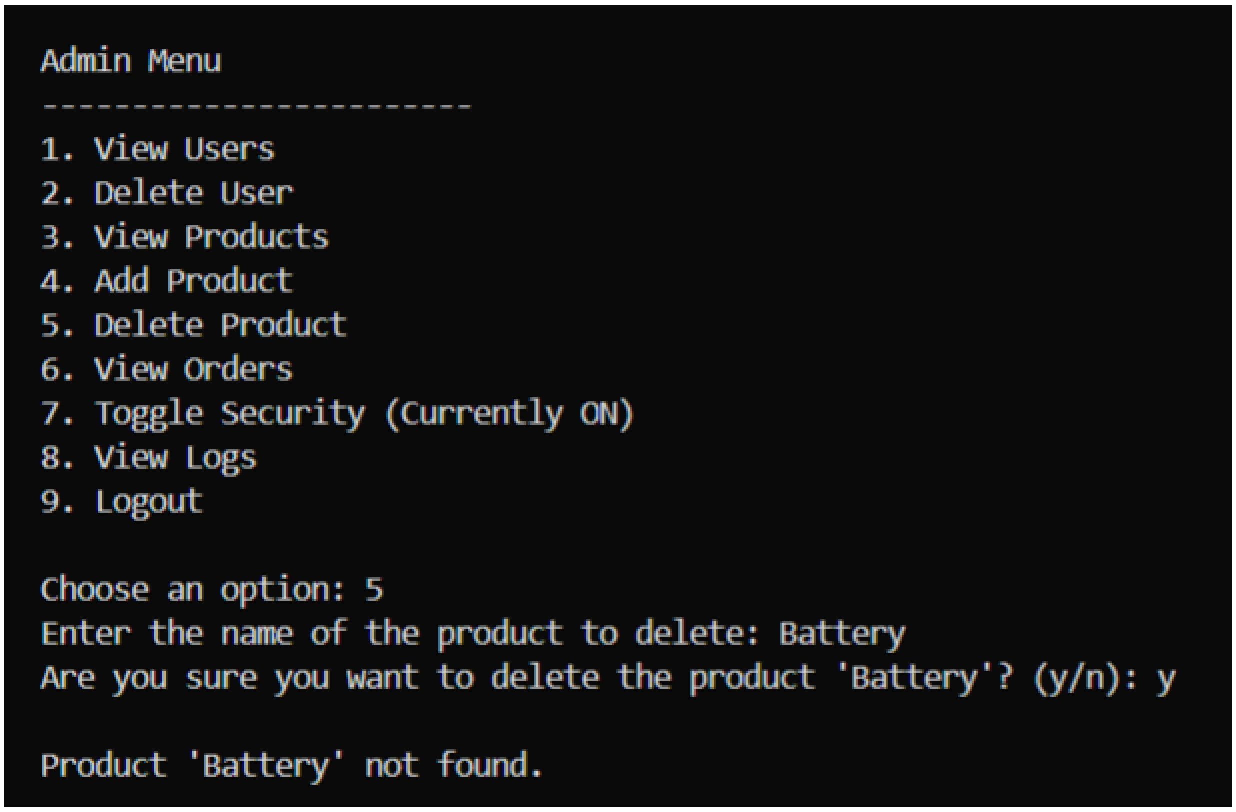
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 4
Enter product name: Laptop
Enter product price (Default is $1.0): 790.99

Product 'Laptop' already exists. Updated price is'$790.99'.
```

Delete Product



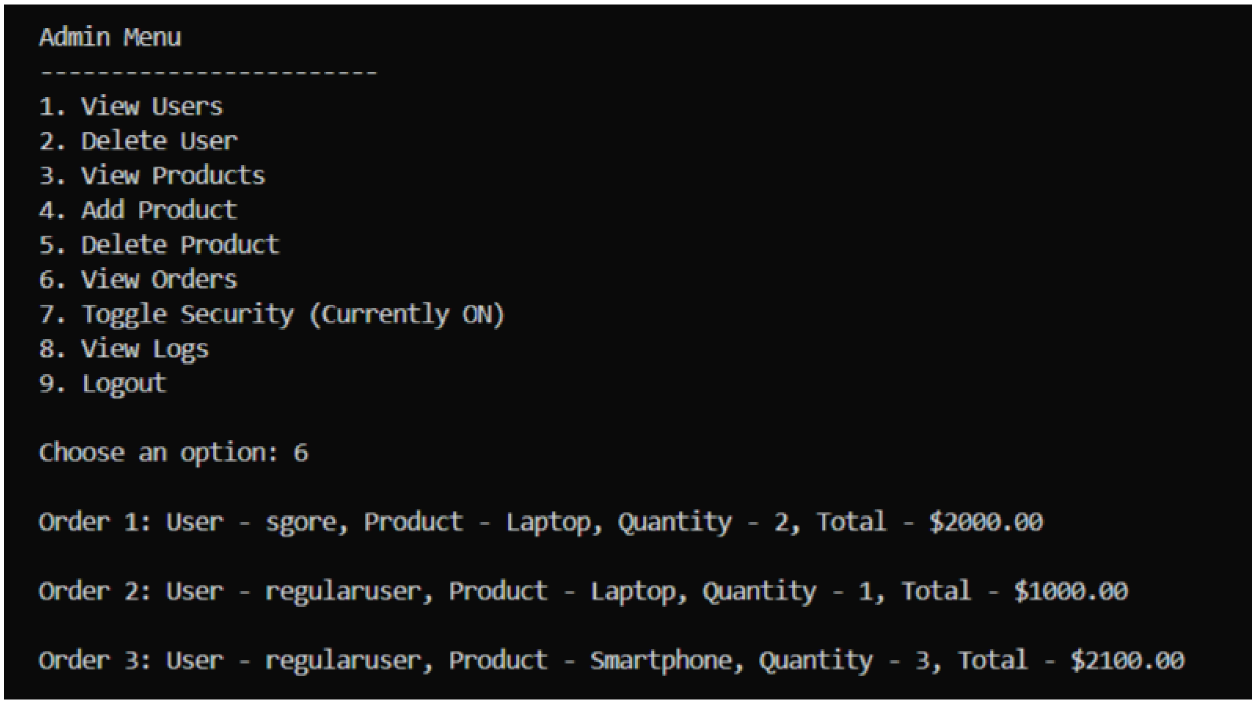
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 5
Enter the name of the product to delete: USB Cable
Are you sure you want to delete the product 'USB Cable'? (y/n): y

Product 'USB Cable' deleted.
```

Delete Product (Non-existent Product)



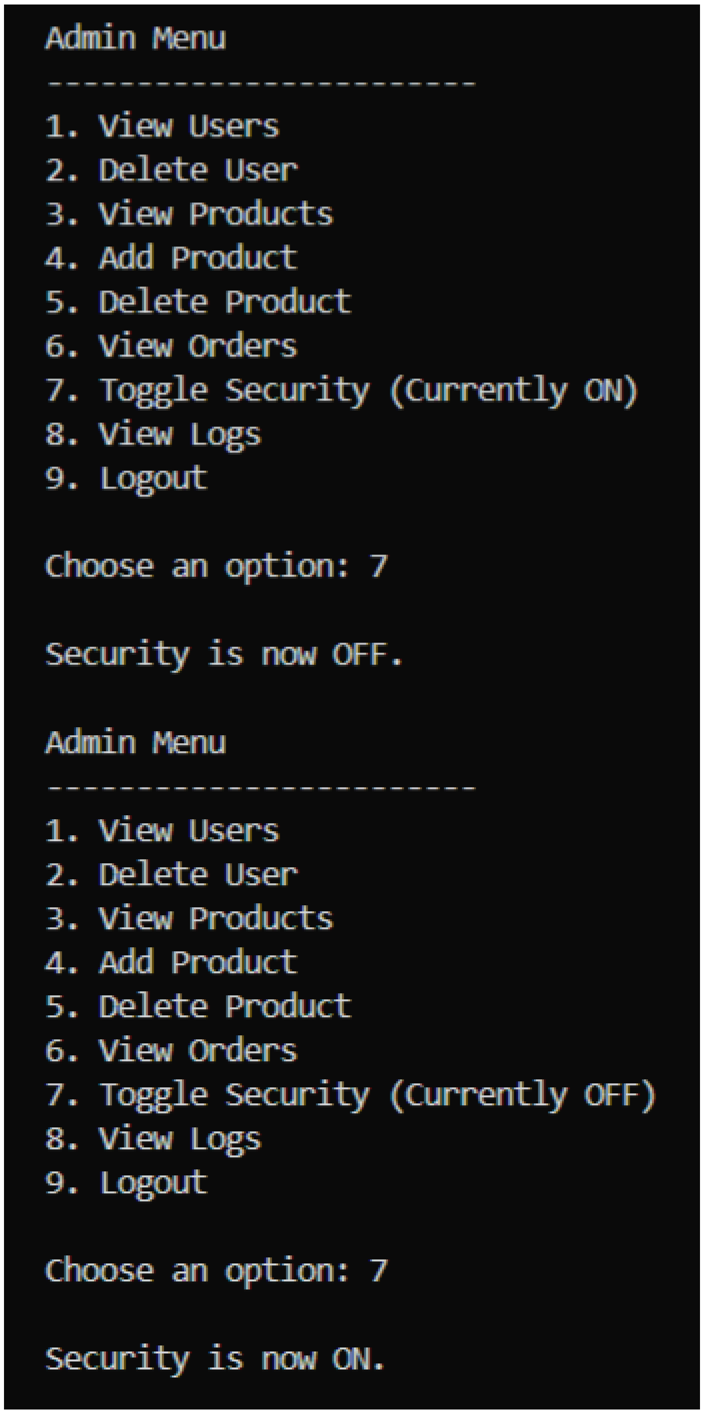
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 5
Enter the name of the product to delete: Battery
Are you sure you want to delete the product 'Battery'? (y/n): y

Product 'Battery' not found.
```

View Orders (Admin User)



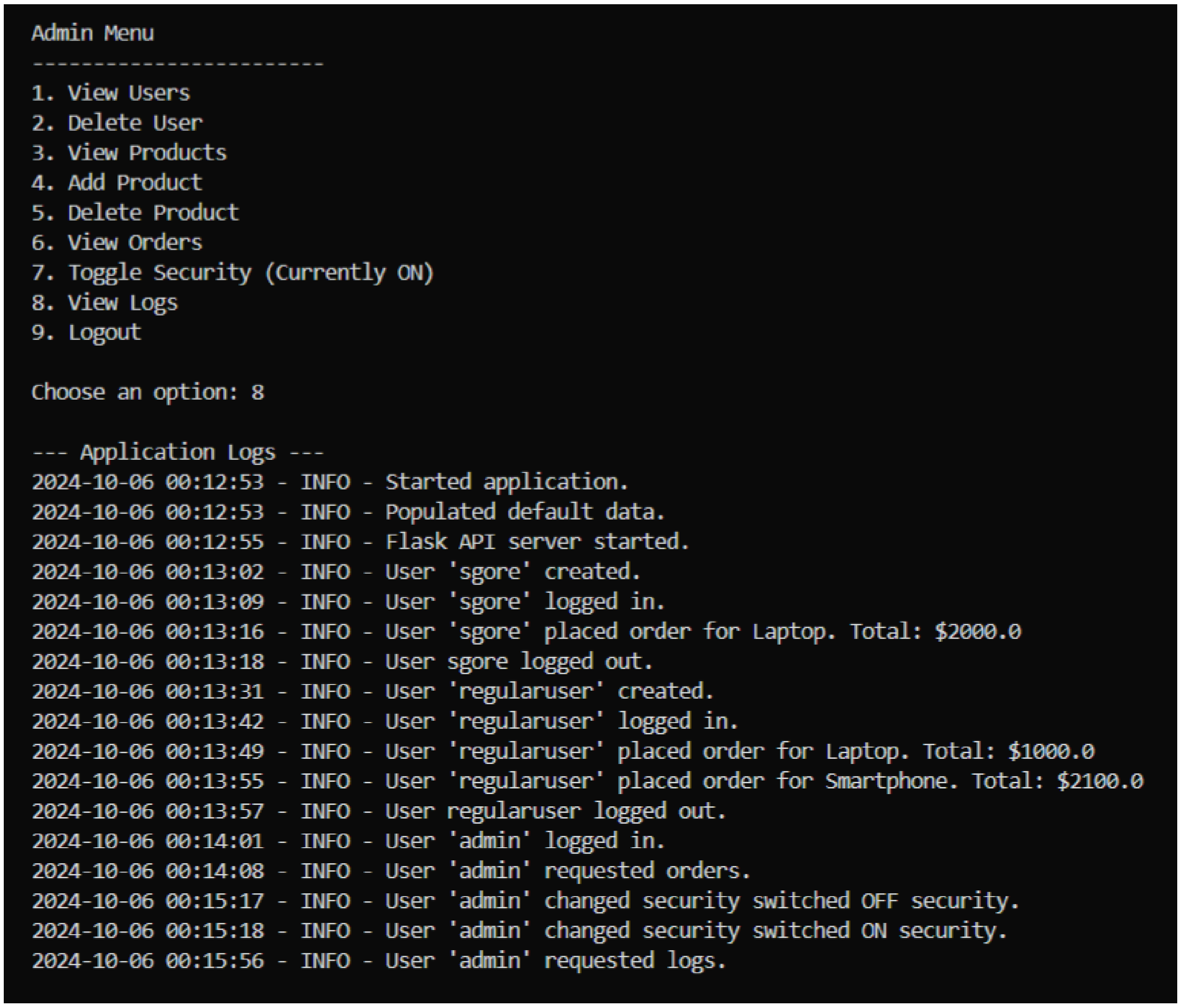
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 6

Order 1: User - sgore, Product - Laptop, Quantity - 2, Total - $2000.00

Order 2: User - regularuser, Product - Laptop, Quantity - 1, Total - $1000.00

Order 3: User - regularuser, Product - Smartphone, Quantity - 3, Total - $2100.00
```

Toggle Security



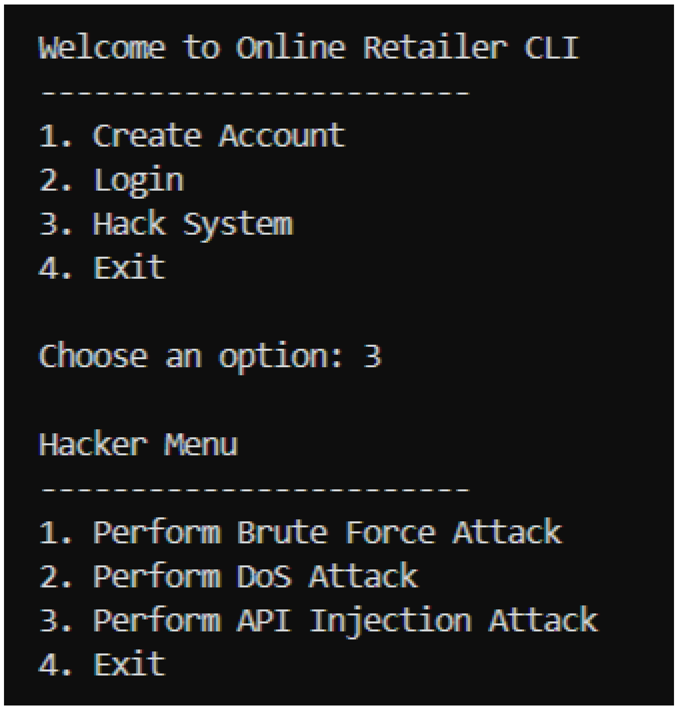
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 7

Security is now OFF.

Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently OFF)
8. View Logs
9. Logout

Choose an option: 7

Security is now ON.
```

View Logs



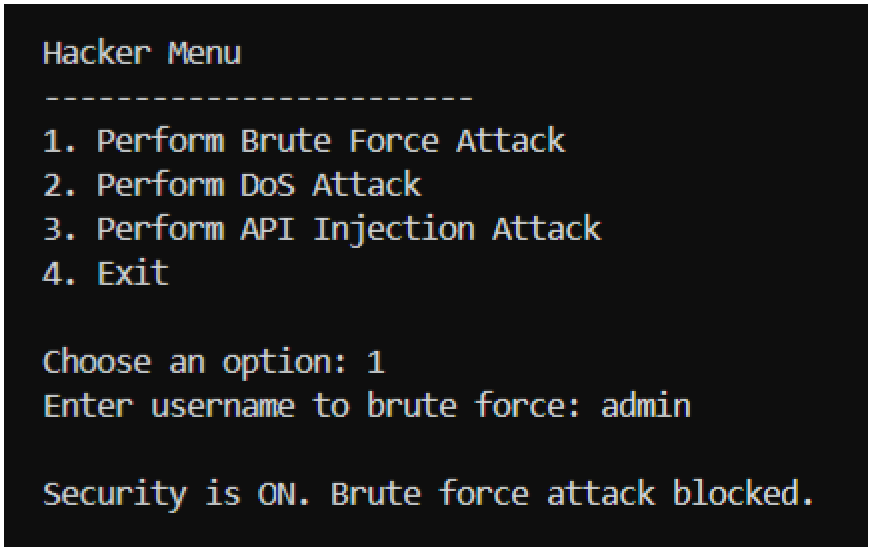
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently ON)
8. View Logs
9. Logout

Choose an option: 8

--- Application Logs ---
2024-10-06 00:12:53 - INFO - Started application.
2024-10-06 00:12:53 - INFO - Populated default data.
2024-10-06 00:12:55 - INFO - Flask API server started.
2024-10-06 00:13:02 - INFO - User 'sgore' created.
2024-10-06 00:13:09 - INFO - User 'sgore' logged in.
2024-10-06 00:13:16 - INFO - User 'sgore' placed order for Laptop. Total: $2000.0
2024-10-06 00:13:18 - INFO - User sgore logged out.
2024-10-06 00:13:31 - INFO - User 'regularuser' created.
2024-10-06 00:13:42 - INFO - User 'regularuser' logged in.
2024-10-06 00:13:49 - INFO - User 'regularuser' placed order for Laptop. Total: $1000.0
2024-10-06 00:13:55 - INFO - User 'regularuser' placed order for Smartphone. Total: $2100.0
2024-10-06 00:13:57 - INFO - User regularuser logged out.
2024-10-06 00:14:01 - INFO - User 'admin' logged in.
2024-10-06 00:14:08 - INFO - User 'admin' requested orders.
2024-10-06 00:15:17 - INFO - User 'admin' changed security switched OFF security.
2024-10-06 00:15:18 - INFO - User 'admin' changed security switched ON security.
2024-10-06 00:15:56 - INFO - User 'admin' requested logs.
```

Hacker Menu



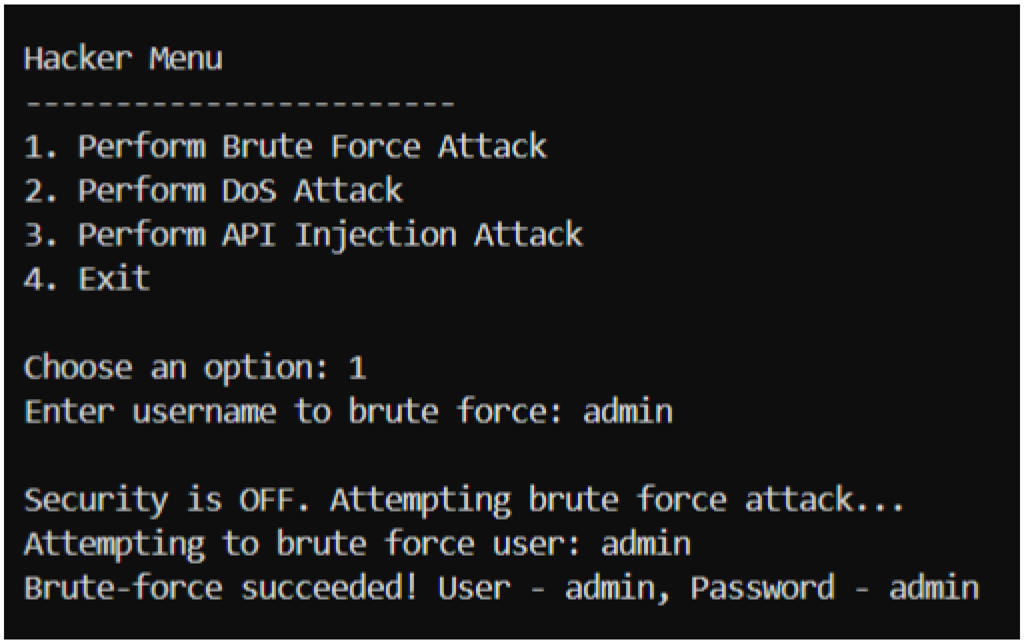
```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 3

Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit
```

Brute Force Attack Simulation (Security is ON)



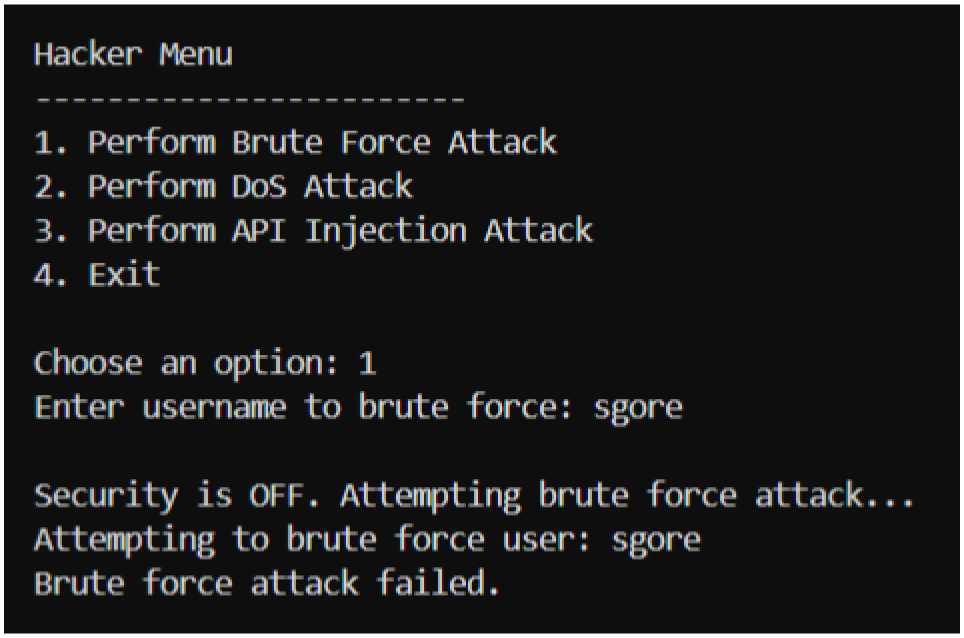
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 1
Enter username to brute force: admin

Security is ON. Brute force attack blocked.
```

Brute Force Attack Simulation (Security is OFF)



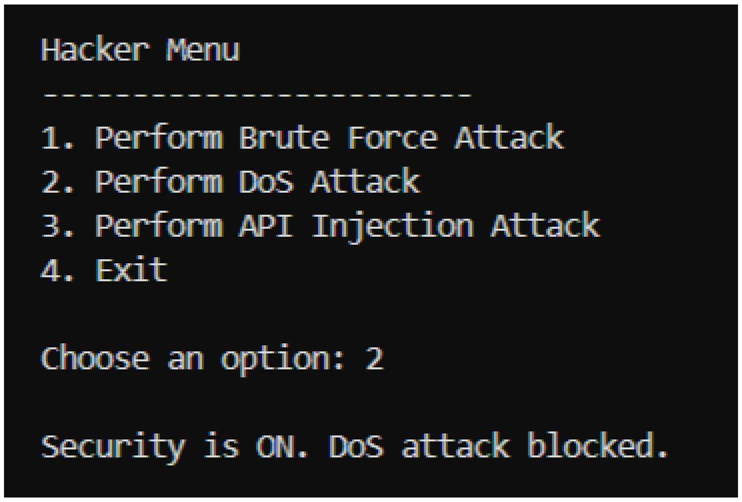
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 1
Enter username to brute force: admin

Security is OFF. Attempting brute force attack...
Attempting to brute force user: admin
Brute-force succeeded! User - admin, Password - admin
```

Brute Force Attack Simulation (Security is OFF - Failed)



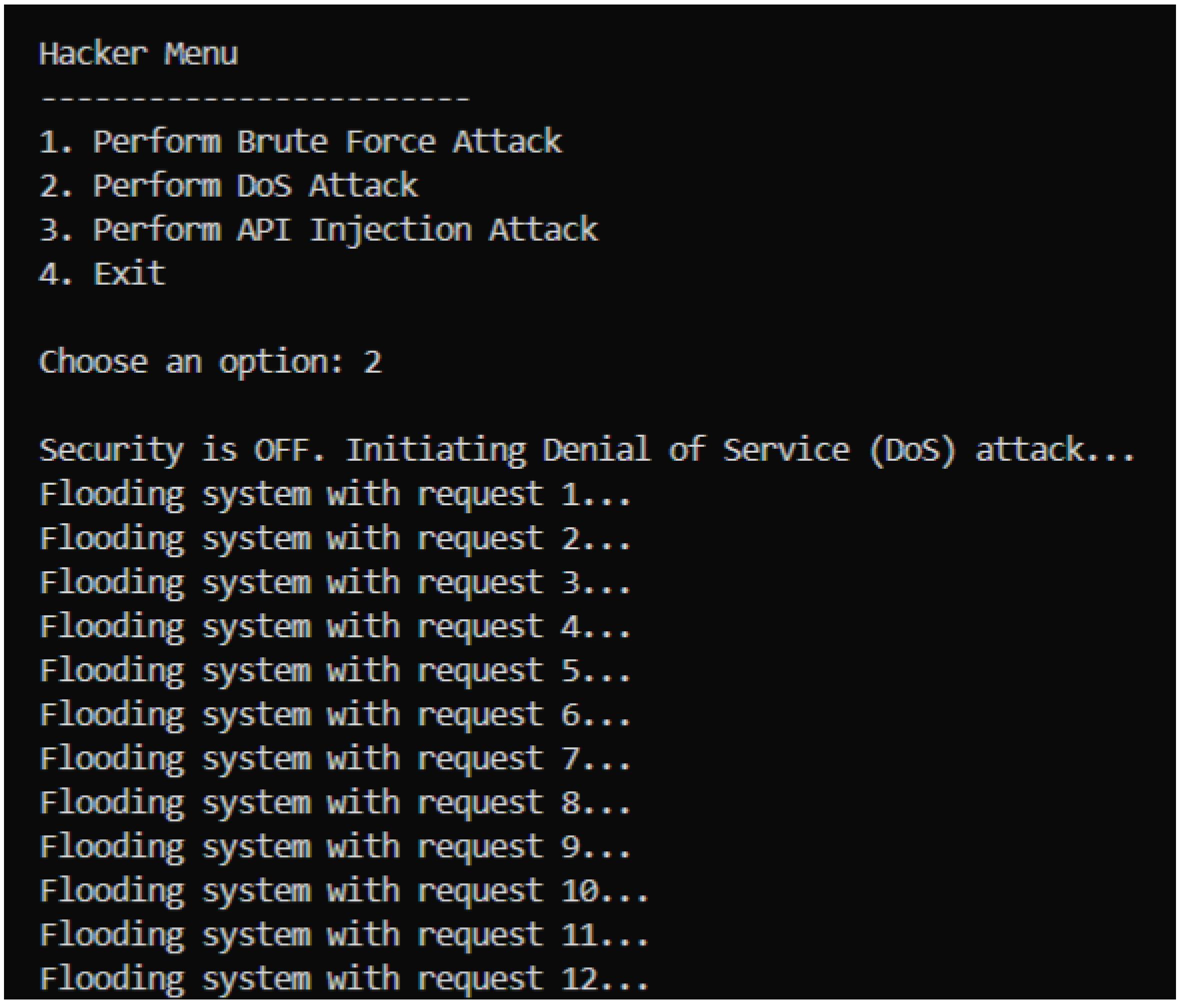
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 1
Enter username to brute force: sgore

Security is OFF. Attempting brute force attack...
Attempting to brute force user: sgore
Brute force attack failed.
```

DoS Attack Simulation (Security is ON)

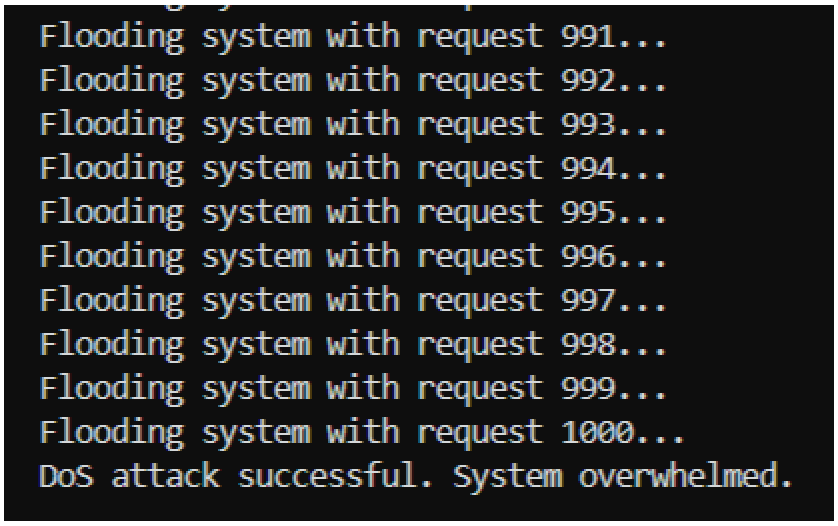


```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 2

Security is ON. DoS attack blocked.
```

DoS Attack Simulation (Security is OFF)

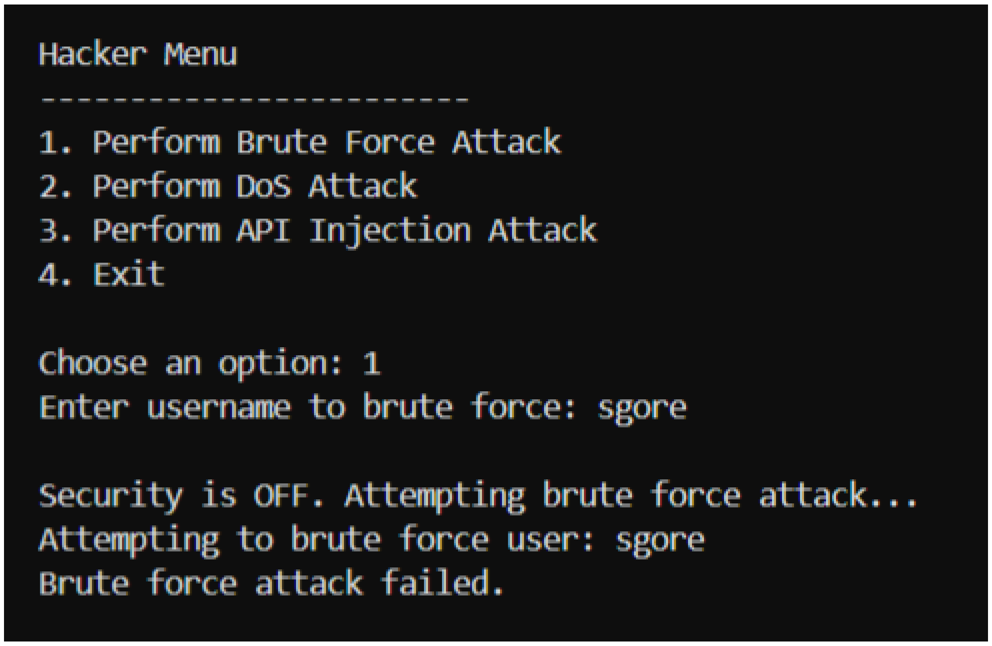


```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 2

Security is OFF. Initiating Denial of Service (DoS) attack...
Flooding system with request 1...
Flooding system with request 2...
Flooding system with request 3...
Flooding system with request 4...
Flooding system with request 5...
Flooding system with request 6...
Flooding system with request 7...
Flooding system with request 8...
Flooding system with request 9...
Flooding system with request 10...
Flooding system with request 11...
Flooding system with request 12...
```

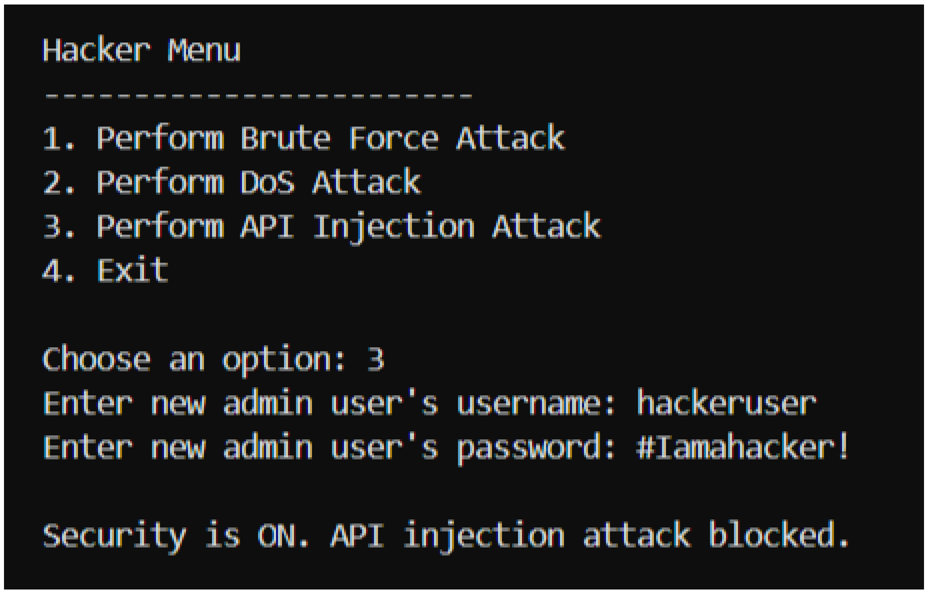


```
Flooding system with request 991...
Flooding system with request 992...
Flooding system with request 993...
Flooding system with request 994...
Flooding system with request 995...
Flooding system with request 996...
Flooding system with request 997...
Flooding system with request 998...
Flooding system with request 999...
Flooding system with request 1000...
DoS attack successful. System overwhelmed.
```

API Injection Attack Simulation (Security is ON)

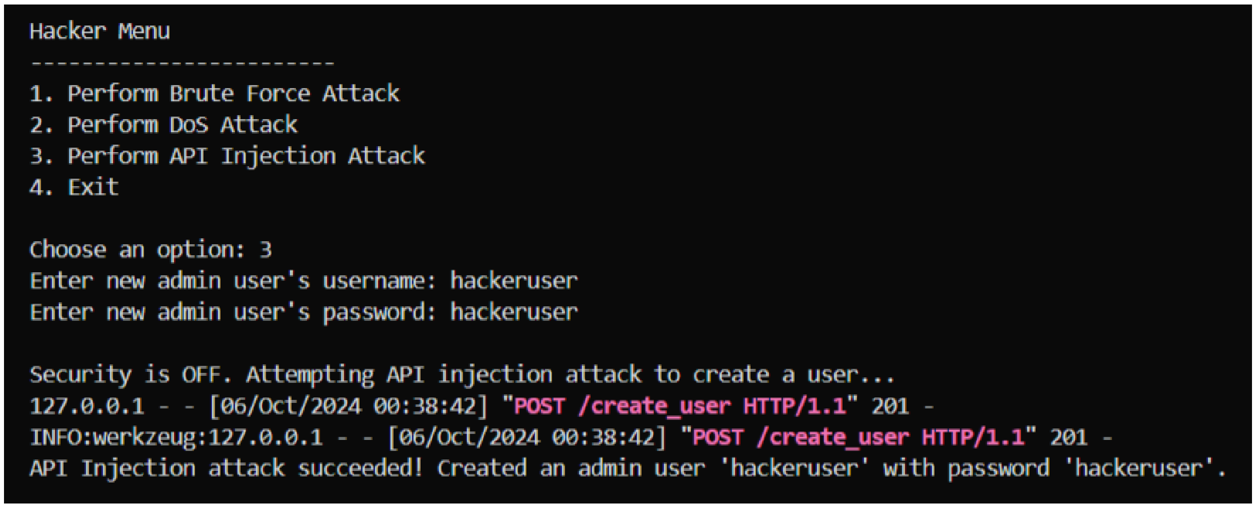


```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 1
Enter username to brute force: sgore

Security is OFF. Attempting brute force attack...
Attempting to brute force user: sgore
Brute force attack failed.
```



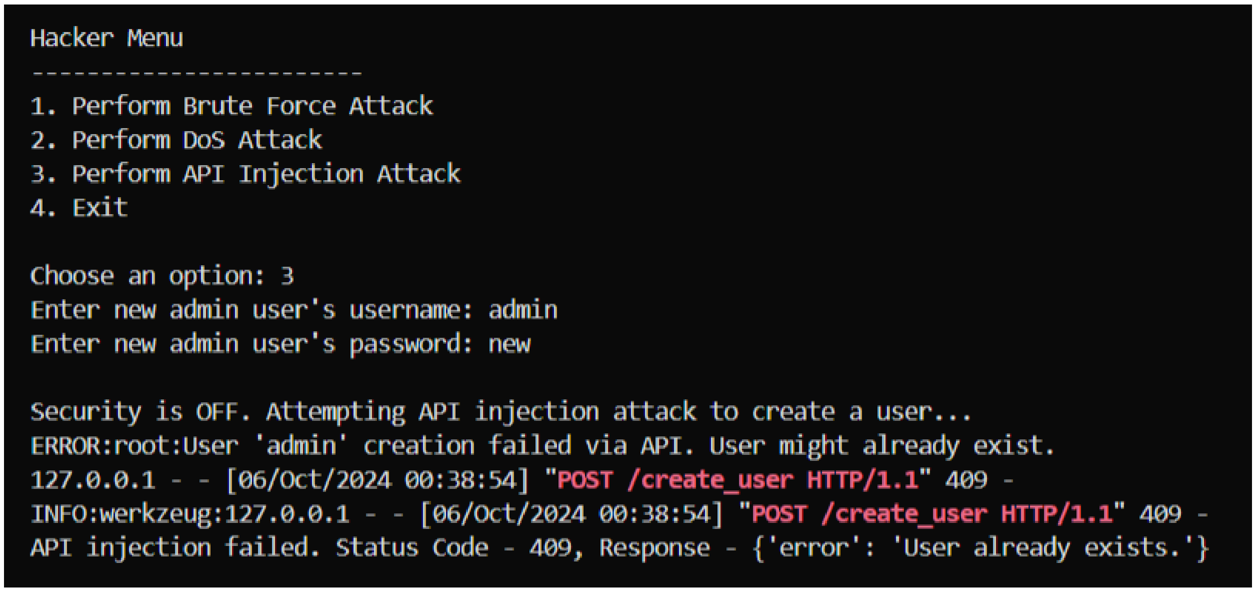
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 3
Enter new admin user's username: hackeruser
Enter new admin user's password: #Iamahacker!

Security is ON. API injection attack blocked.
```

API Injection Attack Simulation (Security is OFF)



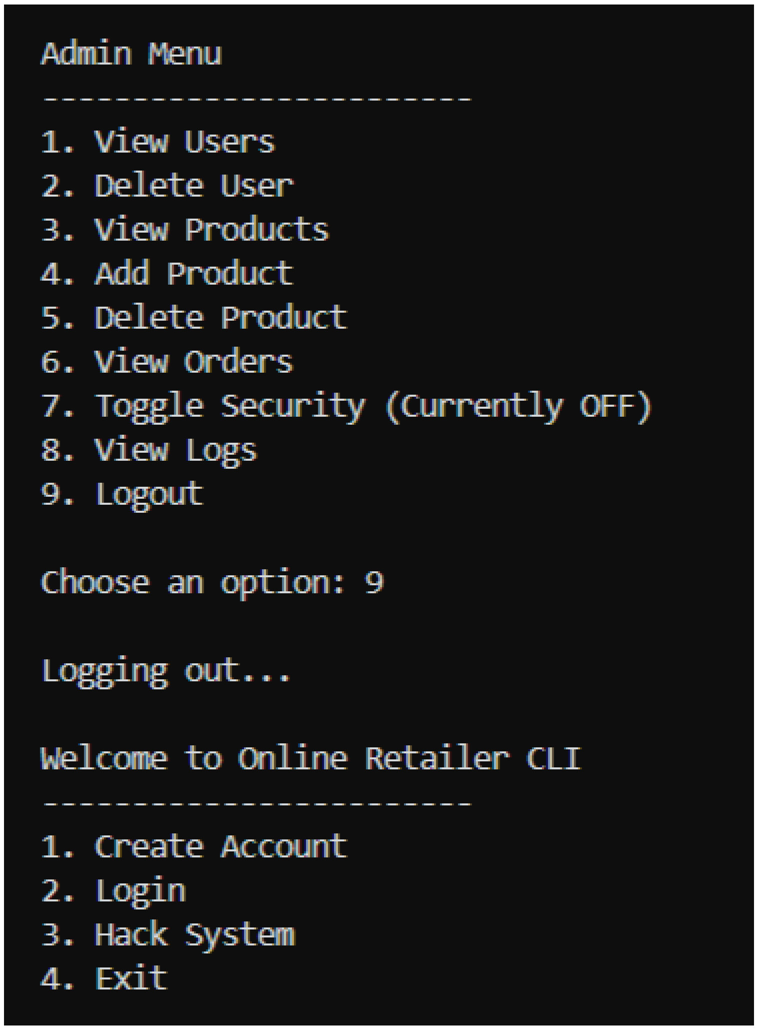
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 3
Enter new admin user's username: hackeruser
Enter new admin user's password: hackeruser

Security is OFF. Attempting API injection attack to create a user...
127.0.0.1 - - [06/Oct/2024 00:38:42] "POST /create_user HTTP/1.1" 201 -
INFO:werkzeug:127.0.0.1 - - [06/Oct/2024 00:38:42] "POST /create_user HTTP/1.1" 201 -
API Injection attack succeeded! Created an admin user 'hackeruser' with password 'hackeruser'.
```

API Injection Attack Simulation (Security is OFF - Failed)



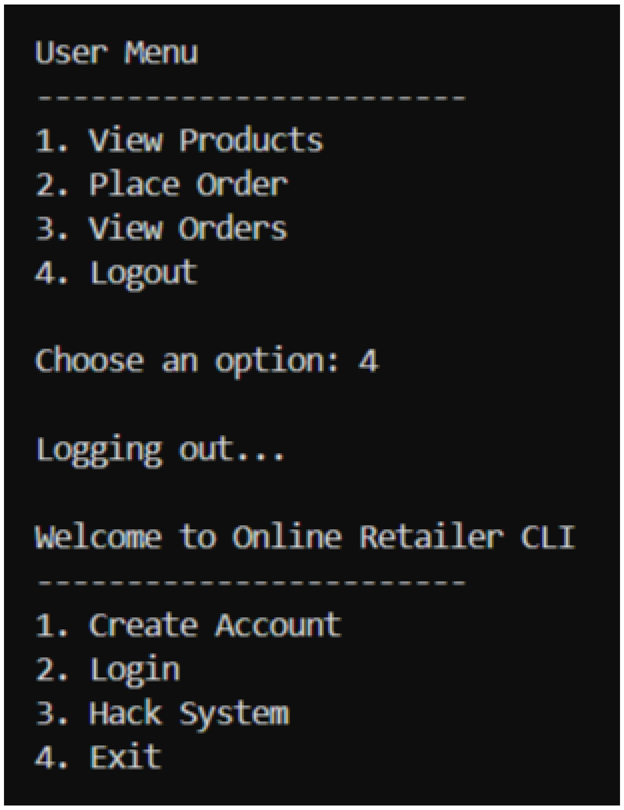
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 3
Enter new admin user's username: admin
Enter new admin user's password: new

Security is OFF. Attempting API injection attack to create a user...
ERROR:root:User 'admin' creation failed via API. User might already exist.
127.0.0.1 - - [06/Oct/2024 00:38:54] "POST /create_user HTTP/1.1" 409 -
INFO:werkzeug:127.0.0.1 - - [06/Oct/2024 00:38:54] "POST /create_user HTTP/1.1" 409 -
API injection failed. Status Code - 409, Response - {'error': 'User already exists.'}
```

Logout (Admin User)



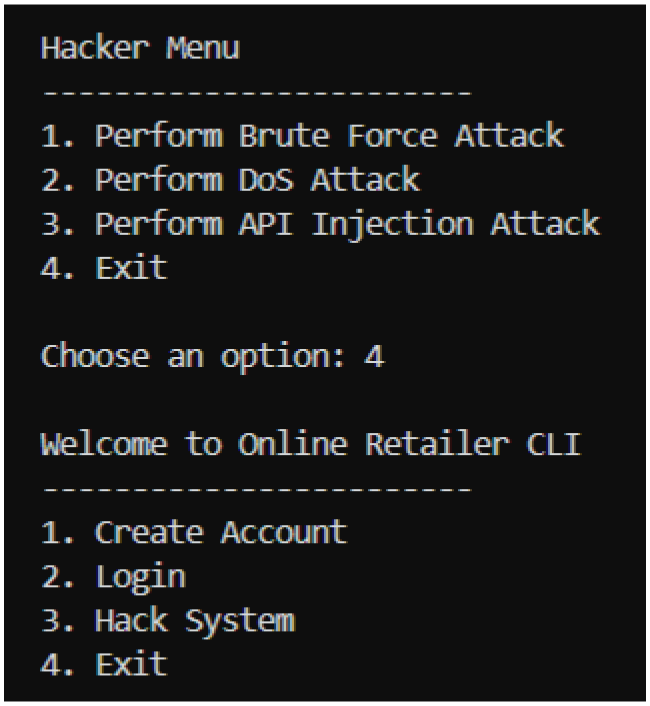
```
Admin Menu
------------------------
1. View Users
2. Delete User
3. View Products
4. Add Product
5. Delete Product
6. View Orders
7. Toggle Security (Currently OFF)
8. View Logs
9. Logout

Choose an option: 9

Logging out...

Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit
```

Logout (Regular User)



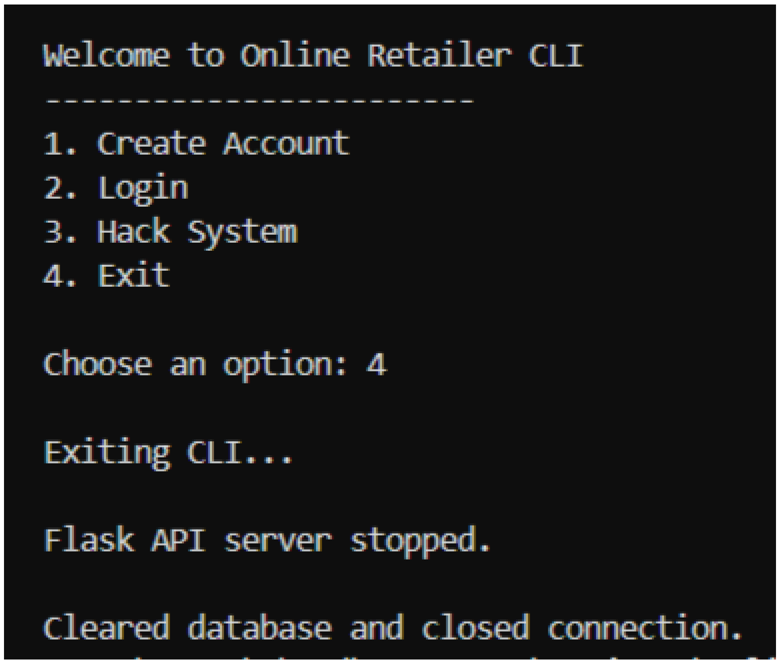
```
User Menu
------------------------
1. View Products
2. Place Order
3. View Orders
4. Logout

Choose an option: 4

Logging out...

Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit
```

Exit Hacker Menu



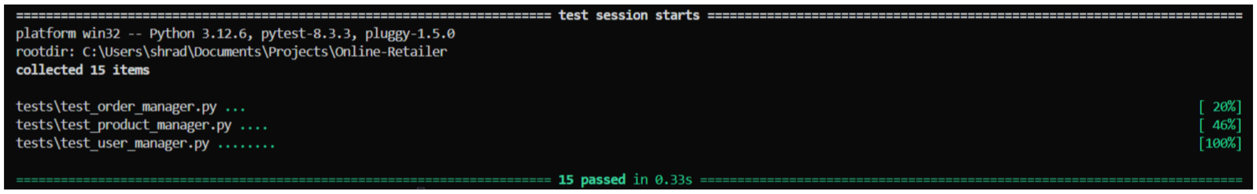
```
Hacker Menu
------------------------
1. Perform Brute Force Attack
2. Perform DoS Attack
3. Perform API Injection Attack
4. Exit

Choose an option: 4

Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit
```

Exit Application



```
Welcome to Online Retailer CLI
------------------------
1. Create Account
2. Login
3. Hack System
4. Exit

Choose an option: 4

Exiting CLI...

Flask API server stopped.

Cleared database and closed connection.
```

Test Results



```
==================================================================== test session starts ====================================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\shrad\Documents\Projects\Online-Retailer
collected 15 items

tests\test_order_manager.py ...                                                                                                                       [ 20%]
tests\test_product_manager.py ....                                                                                                                    [ 46%]
tests\test_user_manager.py ........                                                                                                                   [100%]

==================================================================== 15 passed in 0.33s =====================================================================
```

Conclusion

The developed online retailer system offers a robust and secure platform for users to browse products, place orders, and manage their accounts, all while ensuring that security remains a top priority. The system demonstrates an understanding of cybersecurity threats, giving admins the ability to toggle security features and view system logs, ensuring real-time monitoring and protection against potential attacks. The modular architecture, role-based functionality, and emphasis on security make the online retailer system a reliable, scalable solution suitable for real-world e-commerce environments.

References

GeeksforGeeks (2024) Introduction to SQLite. Available at: https://www.geeksforgeeks.org/introduction-to-sqlite/ [Accessed 5 October 2024].

Fox, L. (2023) Why Should You Use Flask Framework For Web Development. Available at: https://medium.com/@lauren-fox/why-should-you-use-flask-framework-for-web-development-f5a7233e17a6 [Accessed 5 October 2024].

Mezmo (N.D.) Why Logs are the Foundation of Security. Available at: https://www.mezmo.com/learn-observability/why-logs-are-the-foundation-of-security [Accessed 5 October 2024].